Flot de conception automatique pour circuits commutables

Alban Bourge, Olivier Muller, Frédéric Rousseau

Compas'2016 : Parallélisme/ Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Flot de conception automatique pour circuits commutables.

Alban Bourge, Olivier Muller et Frédéric Rousseau

Université Grenoble Alpes, TIMA, F-38000 Grenoble, France
CNRS, TIMA, F-38000 Grenoble, France
{alban.bourge ; olivier.muller ; frederic.rousseau}@imag.fr

Résumé
Les FPGA, ou puces reconfigurables, n'ont pas cessé d'évoluer depuis leur création et sont désormais utilisés dans des systèmes complets (Xilinx Zynq ou Altera Stratix). Malgré tout, il reste de nombreux champs applicatifs desquels ils sont absents, et à tort. Utiliser les FPGA de manière plus intense au sein de systèmes complets est possible, mais il faut pour cela développer les capacités multi-utilisateurs de ces plateformes. Donner la capacité à une application s'exécutant sur un FPGA de se stopper pour, par exemple, laisser s'exécuter d'autres applications jugées prioritaires est particulièrement intéressant. Une telle action est qualifiée de « changement de contexte » (en anglais *context-switch*).
Dans cet article, nous présentons une méthode et un outil permettant de donner cette capacité à des circuits fonctionnant sur cible reconfigurable. Le flot de conception présenté s'appuie sur un logiciel de synthèse de haut niveau et offre automatiquement la capacité de commutation aux circuits synthétisés. Les expériences menées sur un panel de circuits classiques montrent que l'ajout de cette capacité à un coût relativement faible ainsi qu'une rapidité de commutation sans égale dans la littérature.

---

1. Introduction

Le champ applicatif des FPGA est maintenant particulièrement vaste, mais il est possible d'offrir encore plus de débouchés à leur utilisation [10]. Dans cet article, nous allons décrire la brique de base que nous avons développée afin de rendre les tâches matérielles plus aisément contrôlables. Celle-ci consiste à permettre la préemption d'une tâche se déroulant sur une puce reconfigurable ainsi que son redémarrage futur. Cette capacité est connue en tant que « changement de contexte » dans le domaine des microprocesseurs. Les bénéfices qu'offrent une telle capacité son décrits par [23, 20, 8]. On compte parmi eux la possibilité d'intégration de mécanismes dit de reconfiguration [17] ou encore de défragmentation [6]. On simplifiera en admettant qu'un circuit commutable, dont on peut changer le contexte, permet d'ouvrir les tâches matérielles à des scénarios multi-utilisateurs.
Cet article présente un flot de conception de haut niveau (ou HLS pour *High-Level Synthesis*) implantant à des circuits de manière automatique la capacité de changer de contexte. Le second chapitre présente plus en détails nos motivations ainsi que les définitions associées au problème posé. L'état de l'art est abordé dans le chapitre suivant. Le quatrième chapitre présente les contributions de cet article. Celles-ci sont analysées dans le chapitre suivant, et le chapitre final conclut la discussion et expose nos travaux futurs.

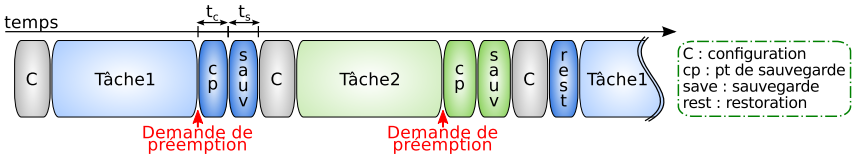

FIGURE 1 – Exemple de changement de contexte entre deux applications

2. Contexte et définitions

Tout au long de cet article, nous allons considérer un système simple servant de base à notre raisonnement. Celui-ci consiste en un processeur associé à un système d'exploitation ainsi que d'un FPGA. Ces deux éléments sont reliés via un bus de communication. En plus de ceux-ci, on peut naturellement considérer qu'une mémoire est connectée sur le même bus.

Tâche matérielle, contexte matériel : Une tâche matérielle est un processus s'exécutant sur une cible matérielle. Dans notre cas, il s'agit d'une tâche s'exécutant sur une puce reconfigurable. En théorie, il est possible d'enregistrer l'état d'une tâche à un instant donné pour la faire repartir depuis le même état, plus tard dans le temps. Cette tâche possède donc un contexte d'exécution propre, par exemple, l'ensemble des registres contenant les données en cours de calcul ainsi que les registres de contrôle. C'est cet ensemble de données, nécessaires au bon redémarrage d'une tâche matérielle, qu'on appelle contexte matériel.

Point de sauvegarde : La notion de point de sauvegarde, ou *checkpoint*, a été initialement développée dans le domaine de la tolérance aux fautes [5, 4]. Notre définition des points de sauvegarde est plus proche de celle de [13] : un instant dans le flot d'exécution d'une tâche matérielle où un changement de contexte est possible.

Changement de contexte matériel : La Figure 1 illustre le déroulement temporel d'un changement de contexte entre deux applications s'exécutant sur un même FPGA. Une première application est lancée après configuration de la puce. Le système envoie une demande de préemption à cette tâche durant son exécution. Celle-ci va devoir atteindre son prochain point de sauvegarde (dans le temps t_c) pour lancer la sauvegarde de son contexte (de durée t_s). Ceci fait, une seconde tâche va pouvoir occuper la ressource, et qui à son tour, suite à une demande de préemption, va laisser la tâche 1 reprendre puis se poursuivre.

3. État de l'art

Le problème à résoudre pour obtenir un mécanisme de changement de contexte efficace consiste à trouver une méthode d'extraction du contexte d'une tâche matérielle. Ce sujet a déjà été abordé dans la littérature et on distingue deux types d'approches pour réaliser l'extraction des données.

Une première façon d'aborder le problème consiste à extraire le contenu de la mémoire de configuration, lorsque cela est possible. On appelle ce type de méthodes *readback*, en rapport avec le mécanisme de relecture du flot de configuration qui est spécifique au constructeur Xilinx [1]. Aucun autre constructeur ne propose de mécanisme similaire. Chez Xilinx, un port spécifique est utilisé pour la configuration ainsi que sa relecture [26, 24]. Bien que les performances de ce type d'approche aient été maintes fois améliorées (voir par exemple [21, 22, 14, 12]), elle présente le principal défaut d'extraire une quantité non négligeable de données non nécessaires au redémarrage de la tâche, comme les informations de routage des différents éléments logiques du circuit dans le FPGA. [12] avance le chiffre de 8% de données utiles.

La seconde méthode consiste à s'affranchir du type de FPGA utilisé et de ne reposer que sur des mécanismes présents dans toutes les puces reconfigurables. Pour ce faire, c'est la description HDL ou netlist de l'application qui va embarquer le mécanisme d'extraction de contexte. Une méthode courante consiste à introduire une chaîne de sérialisation (*scan-chain*) reliant les

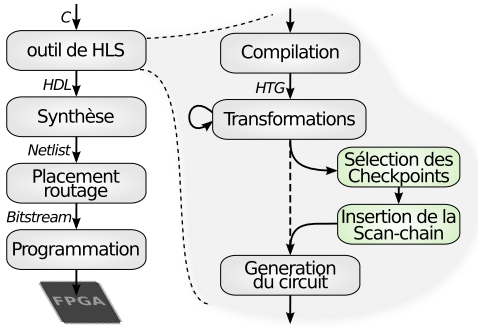

FIGURE 2 – Flot de conception proposé

éléments nécessaires au bon redémarrage de la tâche à l'interface du circuit [25]. [13] présente deux chaînes de sérialisation différentes, en plus d'un mécanisme mettant en relation directe des registres et la mémoire du système. La notion de sélection de points de sauvegarde en vue de réduire le coût d'un changement de contexte est introduite par [16] mais aucune méthode systématique n'est proposée. C'est sur cette seconde méthode d'extraction de contexte que nous allons appuyer nos travaux, en considérant que nous voulons un mécanisme agnostique quant à la cible matérielle utilisée par le concepteur du système complet.

4. Méthode proposée

4.1. Flot de haut niveau

La synthèse de haut niveau consiste à décrire des circuits dans des langages dits de haut niveau, tels que le C ou le C++. Ces langages permettent une description plus concise de l'architecture souhaitée, ainsi qu'un temps de développement réduit sans pour autant devoir négliger les performances [3]. Afin de rendre l'ajout d'un mécanisme de changement de contexte automatique et indépendant du FPGA ciblé, nous avons décidé de développer un outil qui s'inscrit dans un flot de synthèse de haut niveau. En effet, un développeur d'application obtiendra, à partir d'un code d'entrée, un circuit décrit en VHDL contenant en plus la description du mécanisme de changement de contexte. Il reste alors à passer par les phases classiques que sont la synthèse logique, le placement routage ainsi que la programmation pour avoir une application fonctionnelle sur FPGA. Ce flot est représenté plus précisément sur la Figure 2. Les deux contributions proposées dans cet article sont les phases de sélection des points de sauvegarde ainsi que d'ajout de mécanisme d'extraction. Ces deux phases successives s'insèrent dans un flot de HLS afin d'obtenir la fonctionnalité désirée en une seule phase de conception.

4.2. Sélection des points de sauvegarde

Afin de réaliser une opération de changement de contexte, il faut au préalable extraire les données nécessaires au bon redémarrage de la tâche arrêtée. Sélectionner des points de sauvegardes précis, plutôt que d'autoriser l'extraction de contexte à chaque instant, va avoir des effets bénéfiques. En effet, le contexte d'une tâche matérielle est d'un volume non constant dans le temps. À première vue, il consiste essentiellement en tous les éléments mémorisant du circuit exécutant la tâche. Pourtant, à des instants spécifiques de l'exécution, tous les éléments mémorisant ne sont pas nécessaires pour définir l'état de la tâche. En effet, il se peut qu'une partie du contenu des éléments mémorisant ne soit plus utile une fois franchies certaines étapes liées à la tâche exécutée. Par exemple, il n'est pas nécessaire d'extraire la valeur d'un itérateur de boucle une fois celle-ci exécutée. La sélection des points de sauvegarde va consister à trouver, dans le flot d'exécution de la tâche, quels sont les instants dont le volume de données à extraire est faible. On va alors bénéficier de deux avantages en comparaison d'un circuit possédant une

(a) 1 bit de large (b) 8 bits de large

FIGURE 3 – Chaînes de sérialisation

chaîne de sérialisation sur chacun de ses éléments mémorisant :
– la quantité de données à extraire est amoindrie ;
– moins d'éléments mémorisant à équiper signifie un surcout matériel réduit.
La sélection des points de sauvegarde est un problème NP-complet de type « problème de couverture par ensembles » . Une heuristique gloutonne donne une solution acceptable à ce problème. Elle suit une analyse du *Graphe de Tâche Hiérarchique* (HTG [7]) de la tâche d'entrée du flot permettant de calculer une estimation du coût de chaque point de sauvegarde potentiel. Ces deux étapes sont décrites plus précisément dans [2].

4.3. Insertion du mécanisme d'extraction

4.3.1. Mécanisme d'extraction partielle

Le mécanisme d'extraction de contexte consiste à introduire une chaîne de sérialisation entre les éléments mémoires à extraire. Afin de limiter le temps d'extraction ainsi que la quantité des données à extraire, cette chaîne sera spécifique à chaque point de sauvegarde. En effet, chaque *checkpoint* ayant son contexte propre, une chaîne contenant le contexte de chacun des points de sauvegarde aurait peu de sens. Un ensemble de familles de *checkpoints* est donc créé, regroupant les points de sauvegardes partageant un même contexte d'exécution. Chaque famille contient un ou plusieurs *checkpoints*. Afin de ne pas introduire une quantité trop importante de matériel servant à extraire le contexte des différentes familles, on partagera au maximum les chemins logiques existant entre les éléments des différentes chaînes. Le circuit produit possèdera donc autant de chaînes que de familles de points de sauvegardes. Pour créer effectivement les chaînes d'extraction reliant les différents éléments mémorisant entre eux, nous allons devoir différencier deux cas : lorsque l'élément mémorisant à extraire est un registre (accès direct) et lorsqu'il est en mémoire (c'est-à-dire qu'on y accède par une adresse).

4.3.2. Extraction des éléments mémorisant

Dans le cas de l'extraction des registres, la création de la chaîne de sérialisation peut poser de nombreux problèmes, notamment lors du choix de l'ordre des registres dans la chaîne. Nous avons choisi de construire ces chaînes de manière naïve : les registres ne sont pas triés. D'autre part, notre méthode offre la possibilité de construire deux types de chaînes. La Figure 3 représente uniquement les liens créés entre les registres d'une chaîne, les ajouts matériels autres (multiplexeurs) sont omis. La première est une chaîne de sérialisation de un bit de large, cf. Figure 3a. La Figure 3b représente la seconde manière de relier les registres entre eux. Dans ce cas, on crée un lien parallèle de 8 bits (par exemple) de large. On remarque un désavantage de cette seconde méthode : la création de registres dit *inutiles* servant uniquement à la synchronisation interne de la chaîne. À l'inverse l'extraction est plus rapide d'un facteur égal à la largeur de la chaîne comparée à une chaîne d'un bit de large.
L'extraction des mémoires se fait de manière décorrélée par rapport aux registres. En effet, il n'est pas immédiat d'introduire dans une même chaîne d'extraction une mémoire et un registre. Notre choix a été de créer deux mécanismes séparés pour extraire ces deux parties constitutives du contexte matériel d'une tâche comme le montre la Figure 4. L'extraction des registres consiste

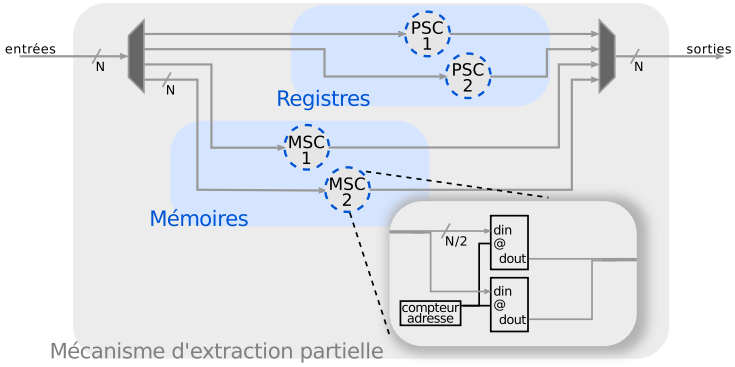

FIGURE 4 – Mécanisme d'extraction partielle, zoom sur les mémoires

en l'association des PSC (pour *Partial Scan-Chain*) alors que l'extraction des mémoires est réalisée par les différentes MSC (*Memory Scan-Chain*). Lorsqu'un changement de contexte est requis, on extrait le contexte relatif aux registres grâce à une PSC spécifique à l'état dans lequel se trouve la tâche matérielle (*checkpoint*). Ceci fait, on extrait le contexte relatif aux mémoires avec une ou plusieurs MSC. On note aussi que pour améliorer la vitesse d'extraction des mémoires, il est possible de mettre plusieurs mémoires côte à côte tant que la somme de la largeur de leurs mots ne dépasse pas la largeur du bus de sortie du contexte.

5. Résultats

Le mécanisme proposé ainsi que la méthode d'insertion à haut niveau ont été développés dans un greffon au logiciel de HLS AUGH [18, 19]. Les résultats produits, des circuits ayant la capacité de changement de contexte, ont été analysés et leur fonctionnalité vérifiée en simulation et sur démonstrateur FPGA.

5.1. Surcout matériel

La solution que nous proposons a un surcout matériel que nous quantifions dans cette partie. La Figure 5 présente les surcouts matériels de 8 circuits issus de la suite de test CHStone [9]. Pour ces 8 circuits, 2×4 totaux relatifs (LUT et FF) sont présentés : le circuit sans mécanisme, le circuit avec une extraction de registre (1 bit de large) uniquement, le circuit avec une extraction de registre (32 bits de large) uniquement, et enfin un circuit avec une extraction des mémoires et des registres de 32 bits de large, ceci afin de bien différencier les différents coûts. On distingue trois groupes de circuits, regroupés en fonction du surcout qu'implique le mécanisme. Pour le premier groupe (AES, IDCT et SHA), le mécanisme peut être considéré comme très peu cher. Les surcouts vont de 3% à 8%. Dans le second groupe, constitué de l'ADPCM, du GSM et du MJPEG, les surcouts sont plus conséquents, mais cela s'explique par la plus grande complexité des circuits générés par l'outil de HLS. Le dernier groupe présente les cas les plus compliqués, qui nécessiteraient d'appliquer des stratégies (dans la sélection des *checkpoints*, dans l'ordonnancement de la chaîne de sérialisation, etc.) plus complexes. Les résultats médiocres du MPEG2 s'expliquent par la méthode de construction naïve des chaînes de sérialisation des registres et l'introduction du mécanisme dans le Blowfish empêche une optimisation faite par ailleurs par l'outil de synthèse logique. Plus généralement, les surcouts sont comparables avec ceux obtenus dans [13], sachant que ce dernier ne propose pas de couvrir l'utilisation des mémoires.

5.2. Comparaison à l'état de l'art

Pour illustrer les gains apportés par notre méthode, le Tableau 1 présente une comparaison avec [11] sur 3 applications de CHStone. Les gains en terme de taille du contexte sont, de

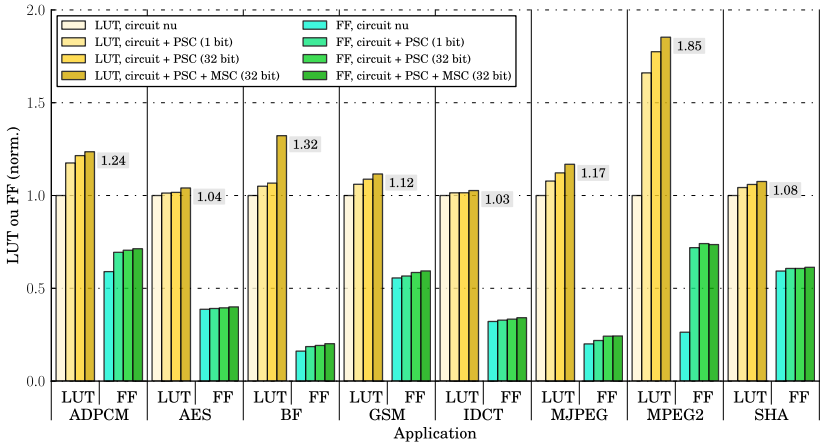

FIGURE 5 – Résultats post synthèse logique réalisés avec ISE 14.7

		[11] *readback* (CPA)	[11] *memory map* (TSAS)	Notre méthode
Taille du contexte	gsm	99,7 kbit	5,37 kbit	5,90 kbit
	idct	49,6 kbit	1,25 kbit	4,20 kbit
	sha	47,5 kbit	1,65 kbit	0,83 kbit
Temps d'extraction	gsm	299 µs	n/a	1,90 µs
	idct	151 µs	266 µs	1,30 µs
	sha	144 µs	351 µs	0,26 µs

TABLE 1 – Comparaison à l'état de l'art

manière prévisible, très nettement en défaveur de la méthode *readback*. Pour une très petite application comme l'IDCT, le contexte de la méthode *readback* atteint approximativement 50 kbit. La méthode *memory map* et la nôtre sont comparables (5,37 kbit vs. 5,90 kbit) car elles permettent l'extraction d'éléments mémoires uniquement. Néanmoins, la méthode consistant à mettre en relation registres et mémoire externe est notablement plus lente. L'extraction du contexte de l'IDCT est effectuée en 266 µs alors que notre méthode permet une extraction en 1,30 µs.

6. Conclusion et travaux futurs

Cet article présente un flot de conception haut niveau qui rend automatique l'insertion d'un mécanisme de changement de contexte dans des circuits. En comparaison des mécanismes pré-existant, il allie rapidité d'extraction du contexte d'une tâche matérielle, légèreté de l'empreinte mémoire à transférer ainsi que souplesse dans le choix potentiel de la cible matérielle. Ces avantages sont obtenus par la combinaison de l'utilisation d'un logiciel de synthèse de haut niveau, permettant la sélection de point de sauvegarde bénéfiques quant au coût de l'extraction, ainsi que de l'ajout d'un mécanisme d'extraction efficace et relativement peu coûteux.
À court terme, il est possible d'étudier plus précisément l'impact de l'ordre des registres dans la chaîne de sérialisation en s'appuyant sur des travaux comme ceux de [27], ou encore d'essayer de profiter de méthodes telles que la "sérialisation gratuite" [15]. Ensuite, il est nécessaire d'intégrer ces travaux dans un projet réussissant à tirer parti de la fonctionnalité proposée, comme de nombreuses références bibliographiques le proposent déjà sans être en possession de la brique de base.

Bibliographie

1. Blodget (B.), James-Roxby (P.), Keller (E.), McMillan (S.) et Sundararajan (P.). – A self-reconfiguring platform. *In : FPL, 2003*, pp. 565–574. – Springer, 2003.
2. Bourge (A.), Muller (O.) et Rousseau (F.). – Automatic high-level hardware checkpoint

selection for reconfigurable systems. – In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23st Annual International Symposium on*, pp. 100–103, May 2015.
3. Coussy (P.) et Morawiec (A.). – *High-level synthesis*. – Springer, 2010.
4. Egwutuoha (I.), Levy (D.), Selic (B.) et Chen (S.). – A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, vol. 65, n3, 2013, pp. 1302–1326.
5. Elnozahy (E. N.), Alvisi (L.), Wang (Y.-M.) et Johnson (D. B.). – A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, vol. 34, n3, 2002, pp. 375–408.
6. Fekete (S. P.), Kamphans (T.), Schweer (N.), Tessars (C.), van der Veen (J. C.), Angermeier (J.), Koch (D.) et Teich (J.). – Dynamic defragmentation of reconfigurable devices. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, n2, 2012, p. 8.
7. Girkar (M.) et Polychronopoulos (C. D.). – The hierarchical task graph as a universal intermediate representation. *International Journal of Parallel Programming*, vol. 22, n5, 1994, pp. 519–551.
8. Guan (N.), Deng (Q.), Gu (Z.), Xu (W.) et Yu (G.). – Schedulability analysis of preemptive and nonpreemptive edf on partial runtime-reconfigurable fpgas. *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, n4, octobre 2008, pp. 56 :1–56 :43.
9. Hara (Y.), Tomiyama (H.), Honda (S.) et Takada (H.). – Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Information and Media Technologies*, vol. 4, n4, 2009, pp. 740–752.
10. Hauck (S.) et DeHon (A.). – *Reconfigurable Computing : The Theory and Practice of FPGA-Based Computation*. – San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2010.
11. Jozwik (K.), Tomiyama (H.), Edahiro (M.), Honda (S.) et Takada (H.). – Comparison of preemption schemes for partially reconfigurable FPGAs. *Embedded Systems Letters, IEEE*, vol. 4, n2, 2012, pp. 45–48.
12. Kalte (H.) et Porrmann (M.). – Context saving and restoring for multitasking in reconfigurable systems. – In *FPL, 2005*. IEEE, 2005.
13. Koch (D.), Haubelt (C.) et Teich (J.). – Efficient hardware checkpointing : concepts, overhead analysis, and implementation. – In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*. ACM, 2007.
14. Levinson (L.), Männer (R.), Sessler (M.) et Simmler (H.). – Preemptive multitasking on fpgas. – In *FCCM, 2000*, pp. 301–302. Citeseer, 2000.
15. Lin (C.-C.), Lee (M. T.-C.), Marek-Sadowska (M.) et Chen (K.-C.). – Cost-free scan : a low-overhead scan path design methodology. – In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pp. 528–533. IEEE Computer Society, 1995.
16. Mignolet (J.-Y.), Nollet (V.), Coene (P.), Verkest (D.), Vernalde (S.) et Lauwereins (R.). – Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. – In *DATE, 2003*, pp. 986–991. IEEE, 2003.
17. Papadimitriou (K.), Dollas (A.) et Hauck (S.). – Performance of partial reconfiguration in fpga systems : A survey and a cost model. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, n4, 2011, p. 36.
18. Prost-Boucle (A.), Muller (O.) et Rousseau (F.). – Méthodologie de génération rapide et automatique d'accélérateurs matériels sous contraintes de ressources : progression itérative et gloutonne. – In *Conférence d'informatique en Parallélisme, Architecture et Système, 2013 (ComPAS'2013)*, oct 2012.
19. Prost-Boucle (A.), Muller (O.) et Rousseau (F.). – Fast and standalone design space exploration for high-level synthesis under resource constraints. *Journal of Systems Architecture*,

vol. 60, 2014, pp. 79–93.

20. Scalera (S. M.) et Vazquez (J. R.). – The design and implementation of a context switching FPGA. – In *FCCM, 1998*, pp. 78–85. IEEE, 1998.
21. Sedcole (P.), Blodget (B.), Becker (T.), Anderson (J.) et Lysaght (P.). – Modular dynamic reconfiguration in virtex fpgas. – In *Computers and Digital Techniques, IEE Proceedings*. IET, 2006.
22. Simmler (H.), Levinson (L.) et Männer (R.). – Multitasking on fpga coprocessors. *In : Field-Programmable Logic and Applications : The Roadmap to Reconfigurable Computing*, pp. 121–130. – Springer, 2000.
23. Trimberger (S.), Carberry (D.), Johnson (A.) et Wong (J.). – A time-multiplexed fpga. – In *FCCM, 1997.*, pp. 22–28. IEEE, 1997.
24. Ullmann (M.), Hübner (M.), Grimm (B.) et Becker (J.). – An fpga run-time system for dynamical on-demand reconfiguration. – In *IPDPS, 2004.*, p. 135. IEEE, 2004.
25. Wheeler (T.), Graham (P.), Nelson (B.) et Hutchings (B.). – Using design-level scan to improve FPGA design observability and controllability for functional verification. – In *FPL, 2001*, pp. 483–492. Springer, 2001.
26. Xilinx. – Partial reconfiguration user guide, 2010. UG702.
27. Zaourar (L.), Kieffer (Y.) et Aktouf (C.). – An innovative methodology for scan chain insertion and analysis at rtl. – In *Test Symposium (ATS), 2011 20th Asian*, pp. 66–71. IEEE, 2011.